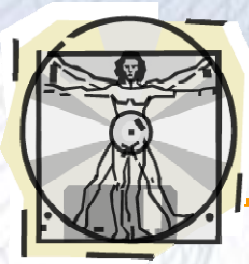
A Methodology for Agent-Oriented Analysis and Design: GAIA

Michael Wooldridge

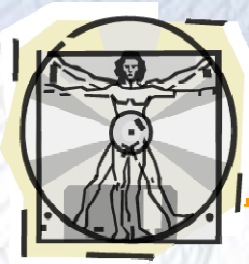
Nicholas R. Jennings

David Kinny



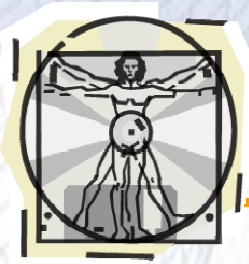
Abstract

- GAIA is a methodology for agent-oriented analysis and design. ← **detailed analysis**
- GAIA methodology is concerned with how a society of agents cooperate to realize the system level goals, and what is required of each individual agent in order to do this.
- The methodology is general. It is applicable to a wide range of multi-agent systems. It deals with both the macro-level (societal) and the micro-level (agent) aspects of systems.
- The methodology is founded on the view of a system as a computational organization consisting of various interacting roles.



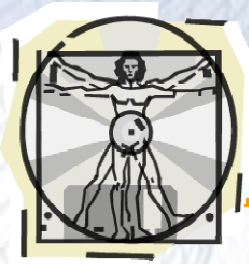
Introduction

- Progress in software engineering over the past two decades has primarily been made through the development of increasingly powerful and natural abstractions with which to model and develop complex systems.
- Procedural abstraction, abstract data types, and, most recently, objects, are all examples of such abstractions.
- It is the authors' belief that agents represent a similar advance in abstraction: they may be used by software developers to more naturally understand, model, and develop an important class of complex distributed systems.



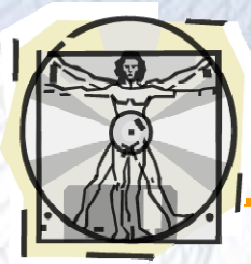
Domain Characteristics

- Agents are coarse-grained computational systems, each making use of significant computational resources (think of each agent as having the resources of a UNIX process.)
- It is assumed that the goal is to obtain a system that maximizes some global quality measure, but which may be suboptimal from the point of view of the system components.
- Agents are heterogeneous, in that different agents may be implemented using different programming languages and techniques.
- The overall system contains a comparatively small number of agents (usually less than 100).

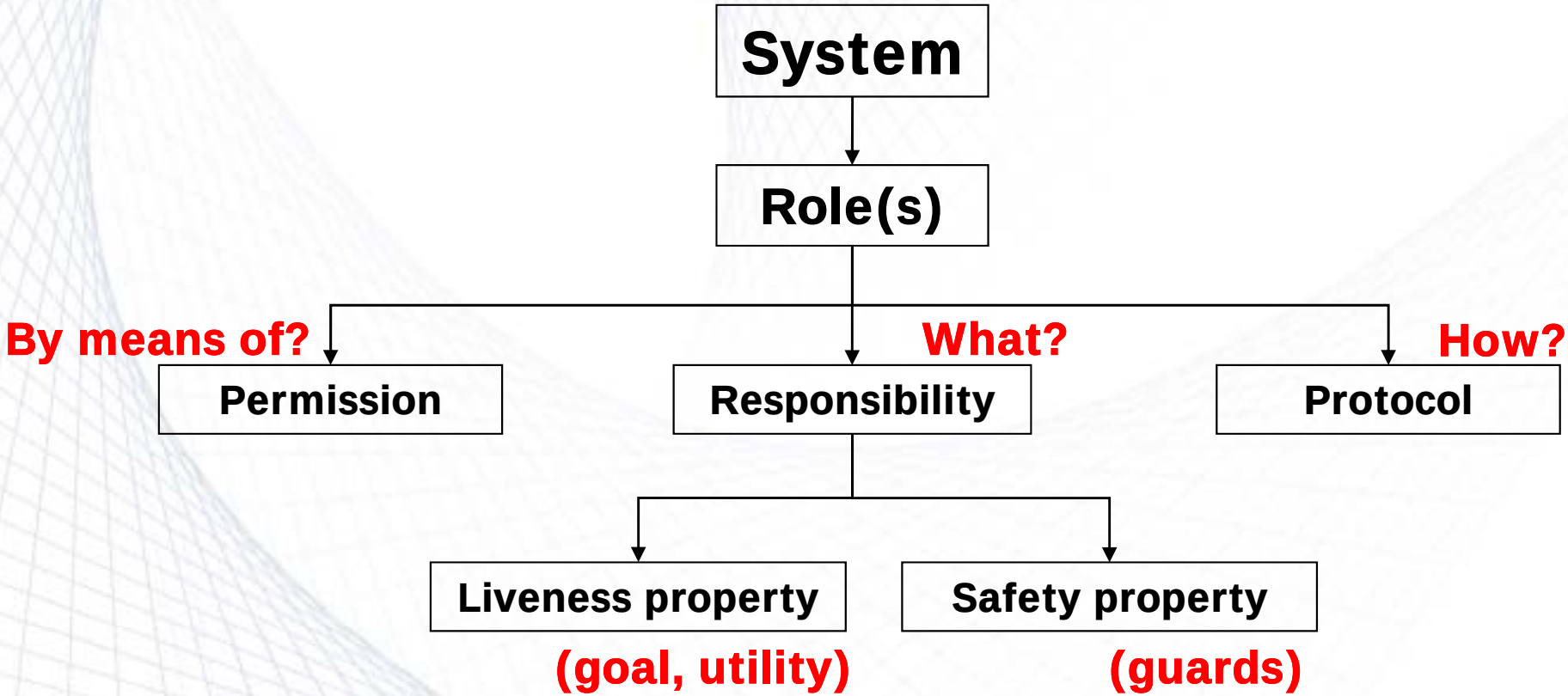


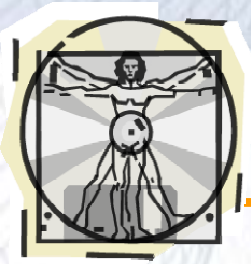
Conceptual Framework

- The methodology is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly.
- In applying the methodology, the analyst moves from abstract to increasingly concrete concepts.
- Each successive move introduces greater implementation bias, and shrinks the space of possible systems that could be implemented to satisfy the original requirements statement.
- The methodology encourages a developer to think of building agent based systems as a process of organizational design.



Analysis Concepts

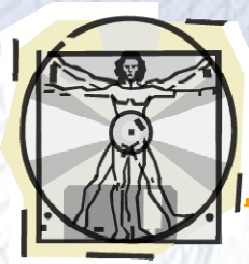




Abstract & Concrete Concepts

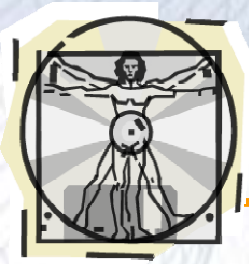
- The concepts used by GAIA are split in two categories:
  - **abstract:** used in the analysis phase, that don't necessary have a correspondent in the implementation of the system
  - **concrete:** concepts that typically have a correspondent in run-time system.

Abstract Concepts	Concrete Concepts
Roles	Agent Types
Permissions	Services
Responsibilities	Acquaintances
Liveness properties	
Safety properties	
Protocols	
Activities	



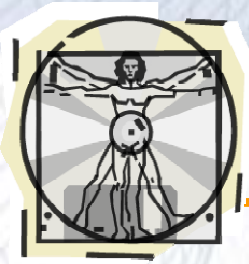
1. System = Society of Agents

- The most abstract entity in the concept hierarchy is the ***system*** which is equivalent to society or organization of agents. That is, we think of an agent-based system as an ***artificial society*** or ***organization***.
- The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: ***roles***.

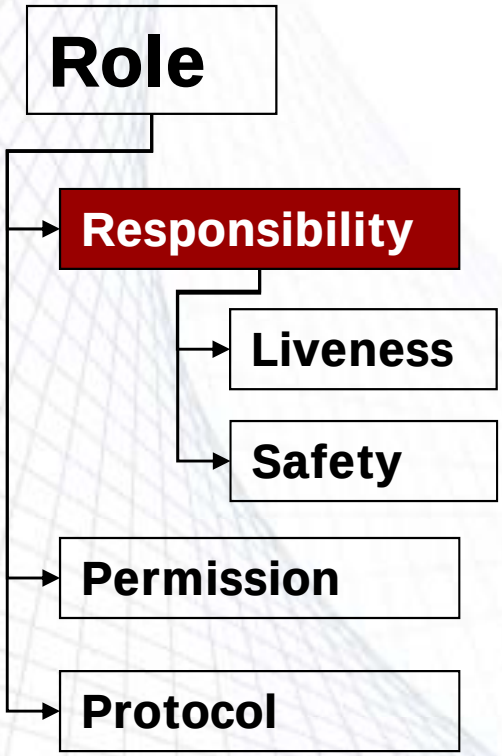


2. Role

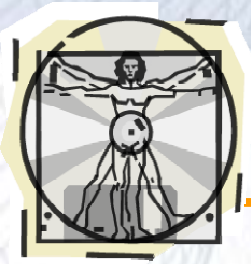
- It may seem strange to think of a computer system as being defined by a set of ***roles***, but the idea is quite natural when adopting an organizational view of the world.
- Consider a human organization such as a typical company. The company has roles such as president, vice president, etc.
- The roles will be instantiated with actual individuals: i.e., individuals take on the role of president, vice president, etc.
- The instantiation is not necessarily static. E.g., throughout the company lifetime, many individuals may take on the role of company president.
- A role is defined by three attributes: ***responsibilities***, ***permissions***, and ***protocols***.



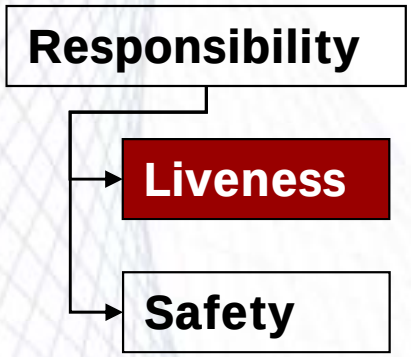
3. Responsibilities (What?)



- Responsibilities determine functionality and are the key attribute associated with a role.
- **Example:** a responsibility associated with the role of company president might be calling the shareholders meeting every year.
- Responsibilities are divided into two types: ***liveness properties*** and ***safety properties***.

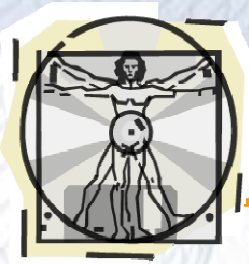


4. Liveness (Goal or Utility)



- ***Liveness properties*** describe what must be achieved by a role under certain environmental conditions.

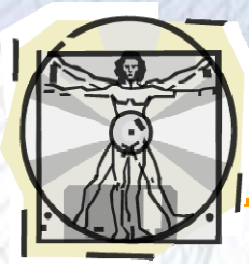
- **Example:** increase profit for "seller" role. Minimize the price while getting the best quality for "buyer" role.



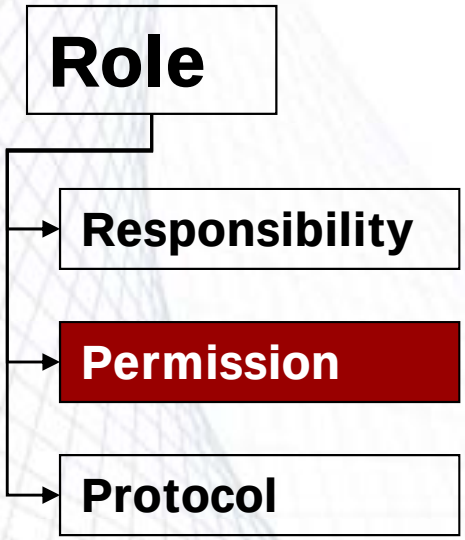
5. Safety (Guards)

- ***Safety properties*** impose a number of constraints on the behavioural states of a role (i.e., that an acceptable state of affairs is maintained across all states of execution).

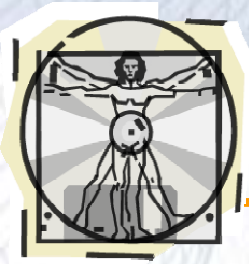
- **Example:** Ensure the liabilities of the company are always less than assets.



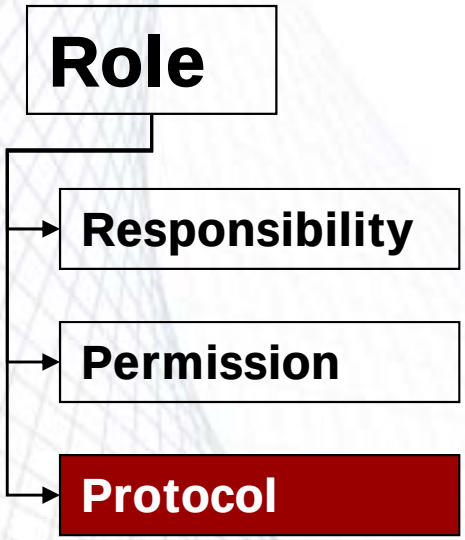
6. Permissions = Rights



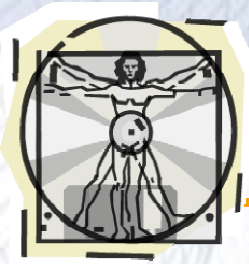
- In order to realise responsibilities, a role is usually associated with a set of permissions.
- Permissions are the rights associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realize its responsibilities.
- In typical systems, permissions tend to be information resources.
- **Examples:**
  - Ability to retrieve a particular information; modify or delete another piece of information.
  - Ability to generate information.



7. Protocols (How?)

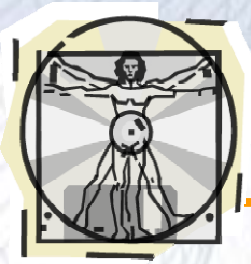


- A role is also identified with a number of protocols, which define the way that a role can interact with the other roles in the system.
- **Example:**
  - A *seller* role might have the protocols *Dutch auction* and *English auction* associated with it.
  - A *hiring* role of the president of a company is to conform to the hiring protocol (policies?).

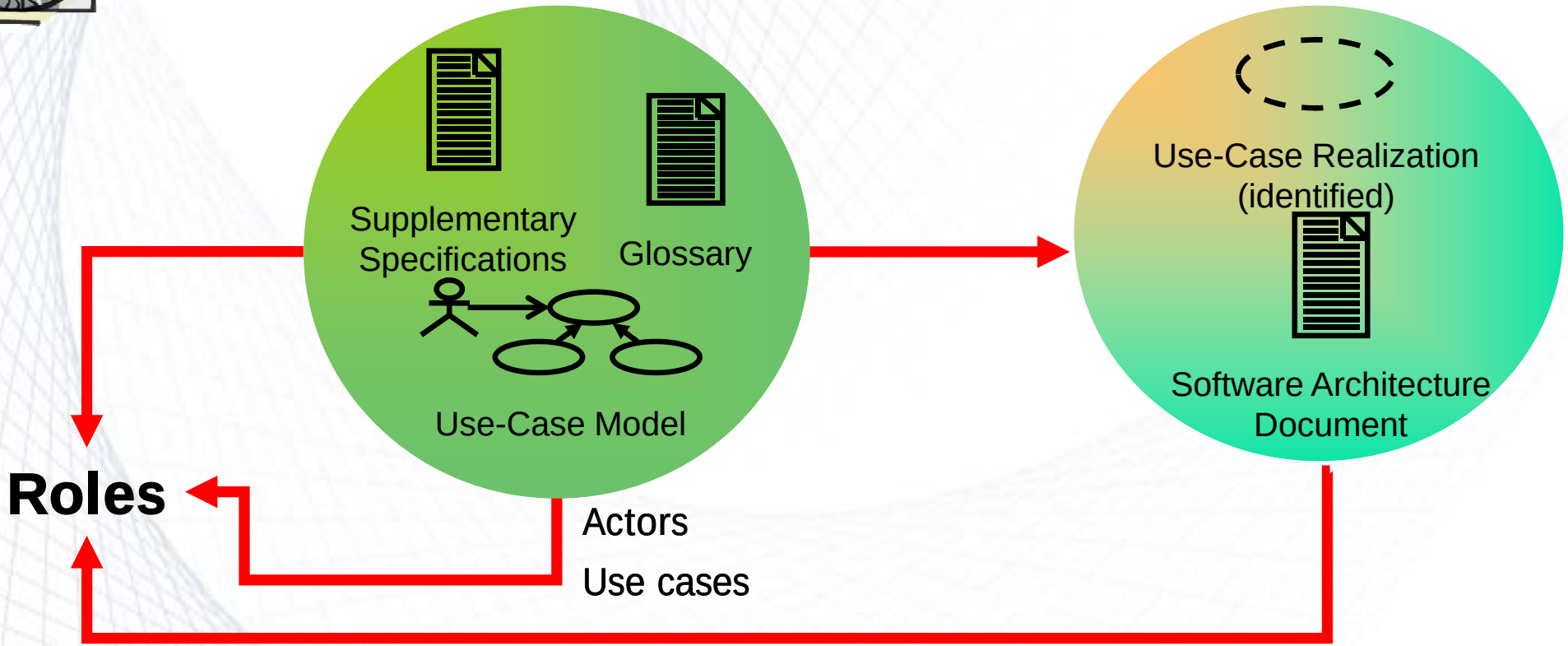


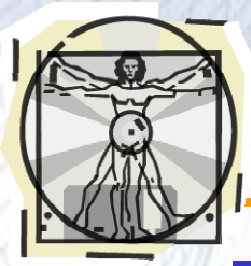
Roles: how to find them?

- Think of possible scenario of system behavior and identify actors in them
- Think of possible must do and must not do actions
- Think of what is needed to do a task and what should be avoided
- Think of possible interactions



How to Find Roles?

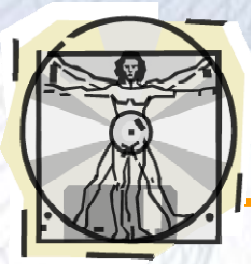




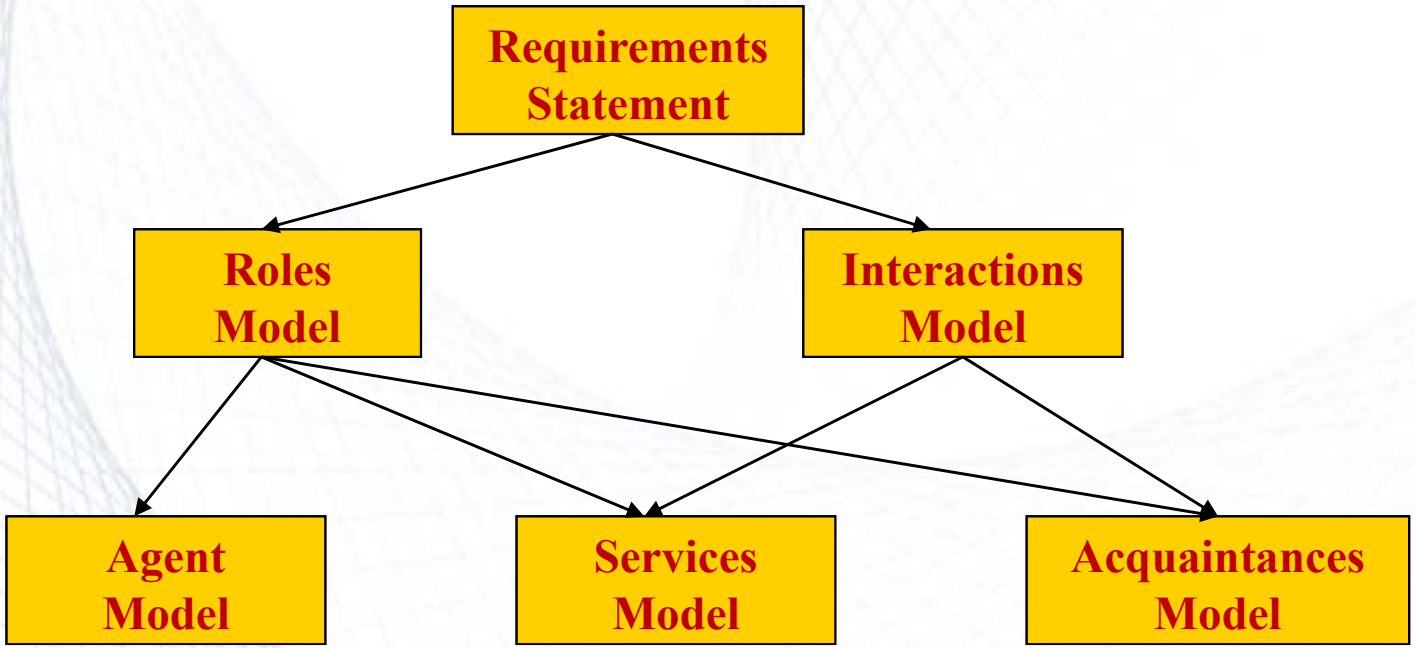
Case of Agile Development /1

- Main source: User stories
- **Example:**

Customer	BSC Clerk
Description	After inspecting/completing a request, can mark the status as held/complete. If "hold", can enter the reason. If "complete", must have all documents attached.
Estimate	4.0 hours
Acceptance Tests	1. Mark as complete with required documents not included. (fail) 2. Mark as complete with required documents included. (pass) 3. Mark as held. (pass)

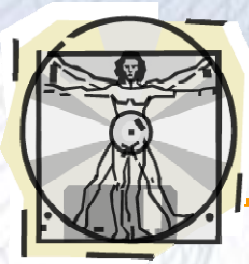


Analysis & Design Models



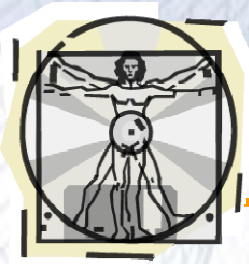
Analysis

Design
aka. Detailed analysis



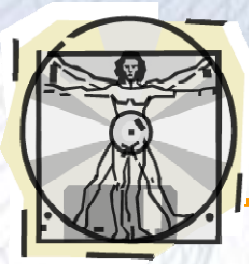
1) Analysis Phase

- The objective of the analysis phase is to develop an understanding of the system and its structure as a ***multi-agent organization***. This understanding is captured in the system organization.
- ***Organization*** is viewed as a collection of roles, that stand in certain relationships to one another, and that take part in systematic, institutionalised patterns of interactions with other roles.
- To define an organization, it suffices to define the roles in the organization, how these roles relate to one another, and how a role can interact with other roles.
- The analysis phase is comprised of two models:
  - ***Role model***
  - ***Interactions model***



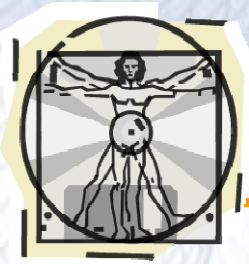
Analysis: Roles Model

- The ***roles model*** identifies the key roles in the system. Here a role can be viewed as an abstract description of an entity's expected function.
- Roles are characterised by two types of attribute:
  - **The permissions/rights associated with the role**
    - A role will have associated with it certain permissions, relating to the type and the amount of resources that can be exploited when carrying out the role. These aspects are captured in an attribute known as the role's permissions.
  - **The responsibilities of the role**
    - A role is created in order to do something. That is, a role has a certain functionality. This functionality is represented by an attribute known as the role's responsibilities.



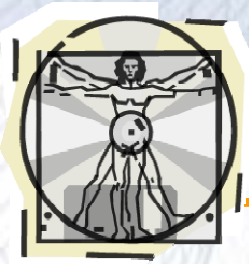
Analysis: Permissions

- The ***permissions*** associated with a role have two aspects:
  - They identify the resources that can legitimately be used to carry out the role, intuitively, they say what can be spent while carrying out the role;
  - They state the resource limits within which the role executor must operate, intuitively, they say what can be spent while carrying out the role.
- In this method, in order to carry out a role, an agent will typically be able to access certain information (or other resources).
- Some roles might generate information; others may need to access a piece of information but not modify it, and others may need to modify the information.



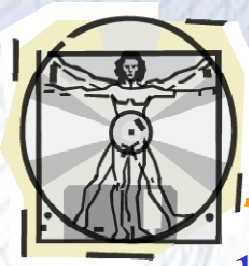
Analysis: Responsibilities

- The functionality of a role is defined by its ***responsibilities***. These responsibilities can be divided into two categories: liveness responsibilities and safety conditions.
- ***Liveness responsibilities*** are so called because they tend to say that something will be done and hence that the agent carrying out the role is still alive.
- Liveness responsibilities tend to follow certain patterns.
- **Example:** The guaranteed response type of achievement goal has the form "request is always followed by a response".
- Liveness expressions define the potential execution trajectories through the various activities and interactions (i.e., over the protocols) associated with the role.



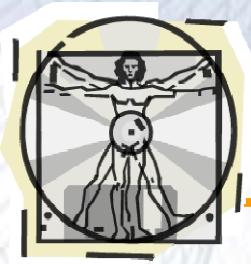
Analysis: Responsibilities

- In many cases, it is insufficient simply to specify the liveness responsibilities of a role. This is because an agent, carrying out a role, will be required to maintain certain invariants while executing.
- **Example:** We might require that a particular agent taking part in an electronic commerce application never spends more money than it has been allocated.
- These invariants are called ***safety conditions***, because they usually relate to the absence of some undesirable condition arising.
- Safety requirements are specified by means of a list of predicates. These predicates are typically expressed over the variables listed in a role's permissions attribute.



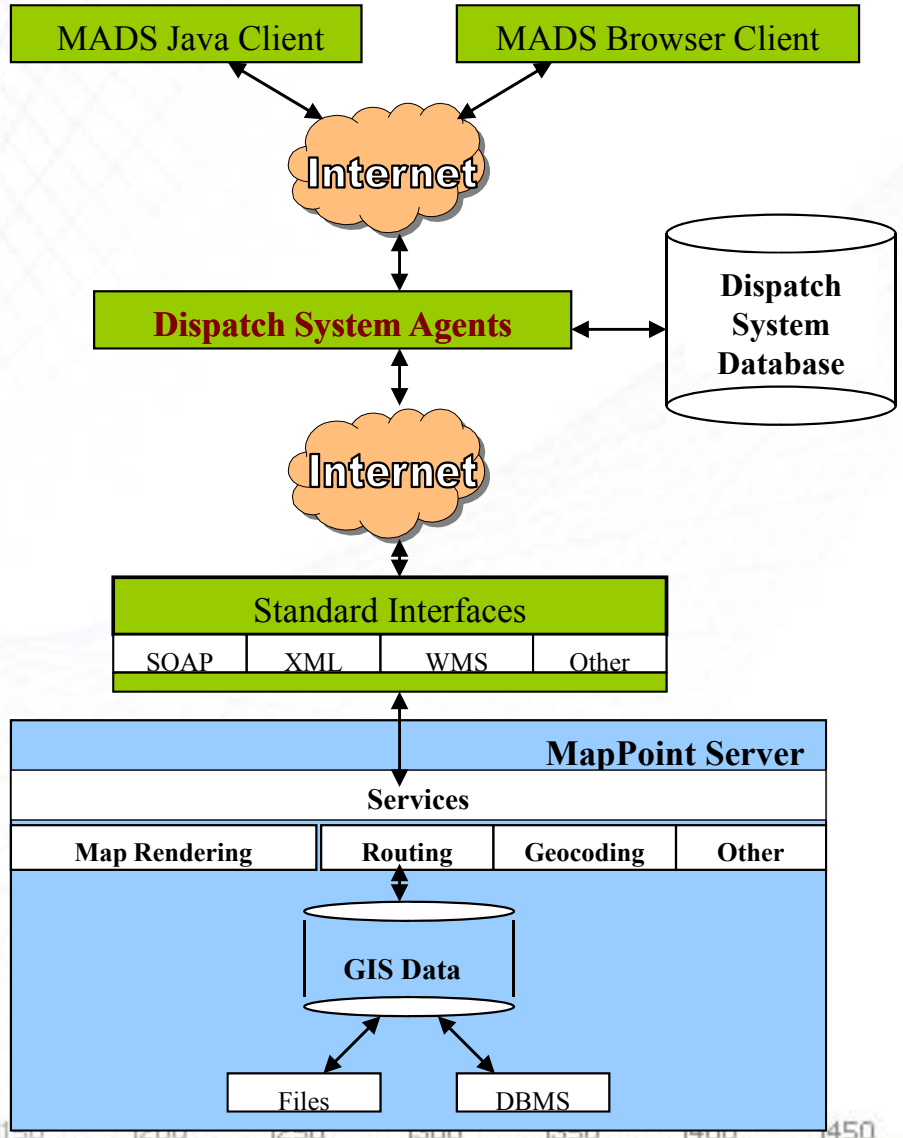
GAIA: Analysis Process

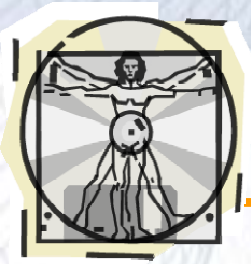
1. Identify the ***roles*** in the system.
   - **Output:** A prototypical roles model, a list of the key roles that occur in the system, each with an informal, unelaborated
2. For each role, identify and document the associated protocols. Protocols are the patterns of ***interaction*** that occur in the system between the various roles.
   - **Output:** An interaction model, which captures the recurring patterns of inter-role interaction.
3. Using the protocol model as a basis, elaborate the roles model.
   - **Output:** A fully elaborated roles model, which documents the key roles occurring in the system, their permissions and responsibilities, and the protocols in which they take part.
4. Iterate stages (1)- (3).



Example System

- Multi-Agent Dispatch System (MADS) provides Road-side assistance system for a fleet of vehicles





Example: MADS System

- **Role:** ***Geocode***

  Description: wraps the MapPoint geocoding Web Services and it queries MapPoint for geocodes.

- **Role:** ***RenderMap***

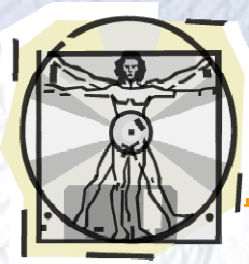
  Description: It wraps the MapPoint map rendering Web Services and it queries MapPoint for maps.

- **Role:** ***Route***

  Description: It wraps the MapPoint routing Web Services and it queries MapPoint for routes.

- **Role:** ***PersonalAssistant***

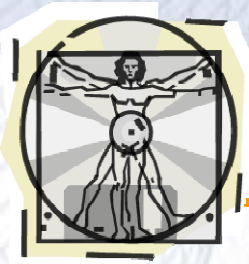
  Description: It acts on behalf of a system user. It gets the geocode information, maps and routes, and it calculates distances between the breakdown and available service providers.



Example (cont'd)

Permissions for *RenderMap*:

- reads
  - *MapPointWebServices*
  - *newMap* // true or false
- changes
  - *notifyUser* // true or false



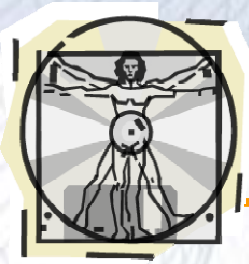
Example (cont'd)

Liveness Property:

- When the *newMap* is true, get a new map.
- When the new map is ready, notify the user.

RENDERMAP = (RenderMap)

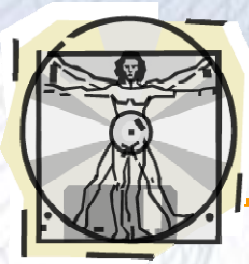
RENDERMAP = RequestMap.QueryMapPoint.RespondMap



Example (cont'd)

Safety Property:

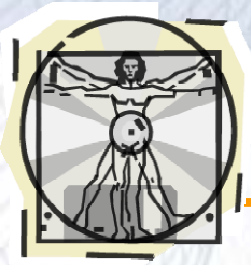
- Safety properties are specifies using a list of predicates in relationship with the variables listed in the role's permissions attribute.
- For the *RenderMap* role, a successful connection with MapPoint Web Services is the safety property.



Role Schema

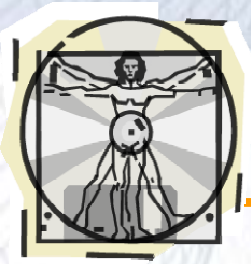
- GAIA roles model is composed of a set of role schemata for each role in the system with the following template

Role Schema		*name of role*
Description		*short description of the role*
Protocols and Activities		*protocols and activities in which the role plays a part*
Permissions		*rights associated with the role*
Responsibilities	Liveness	*liveness responsibilities/properties*
	Safety	*safety responsibilities/properties*



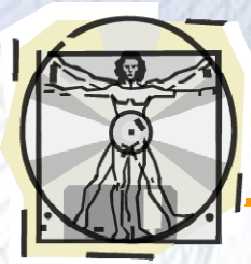
Example: RenderMap

Role Schema		**RenderMap**
Description		It wraps the MapPoint map rendering Web Services and it queries MapPoint for maps.
Protocols and Activities		QueryMapPoint, RequestMap, RespondMap
Permissions		reads MapPoint Web Services
Responsibilities	Liveness	RENDERMAP = (RenderMap) RENDERMAP = RequestMap.QueryMapPoint.RespondMap
	Safety	a successful connection with MapPoint Web Services



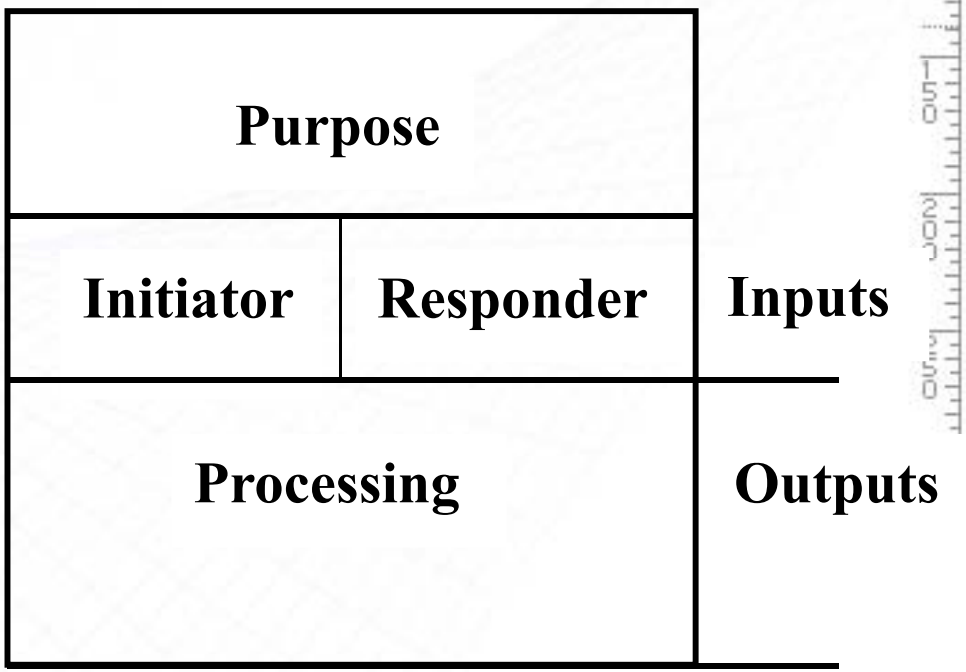
Analysis: Interaction Model

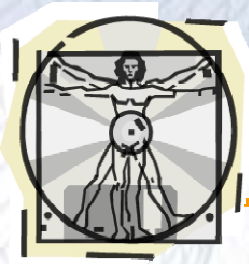
- There are dependencies and relationships between the various roles in a multi-agent organisation. Interactions need to be captured and represented in the analysis phase.
- This model consists of a set of ***protocol definitions (mechanisms)***, one for each type of inter-role interaction.
- A protocol can be viewed as a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps.
- A single protocol definition will give rise to a number of message interchanges in the run time system.
- **Example:** English auction protocol. This involves multiple roles (sellers and bidders) and many potential patterns of interchange (price announcements and corresponding bids). However at the analysis stage, such precise instantiation details are not necessary.



Analysis: Interaction Model

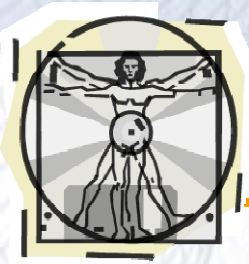
- Protocol definitions consist of the following set of attributes:
  - ***Purpose:*** brief description of the nature of the interaction (e.g. Information request, schedule activity and assign task;
  - ***Initiator:*** the role(s) responsible for starting the interaction;
  - ***Responder:*** the role(s) with which the initiator interacts;
  - ***Inputs:*** information used by the role initiator while enacting the protocol;
  - ***Outputs:*** information supplied by/to the protocol responder during interaction;
  - ***Processing:*** brief description of any processing the protocol initiator performs during the course of the interaction.





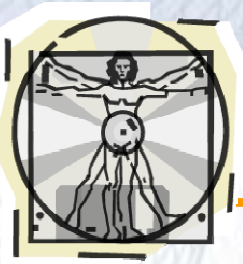
Example: Interaction Model

Protocol	`RequestGeocode`	`RequestMap`	`RequestRoute`
Purpose/ parameters	Request to geocode a location. The response includes the Latitude and Longitude of the location	Request a map containing a list of location. The response include the URL to the map that displays the locations	Request a route between two locations. The response includes the driving direction between the locations
Initiator(s)	PersonalAssistant	PersonalAssistant	PersonalAssistant
Receiver(s)	Geocode	RenderMap	Route
Responding Protocol	RespondGeocode	RespondMap	RespondRoute

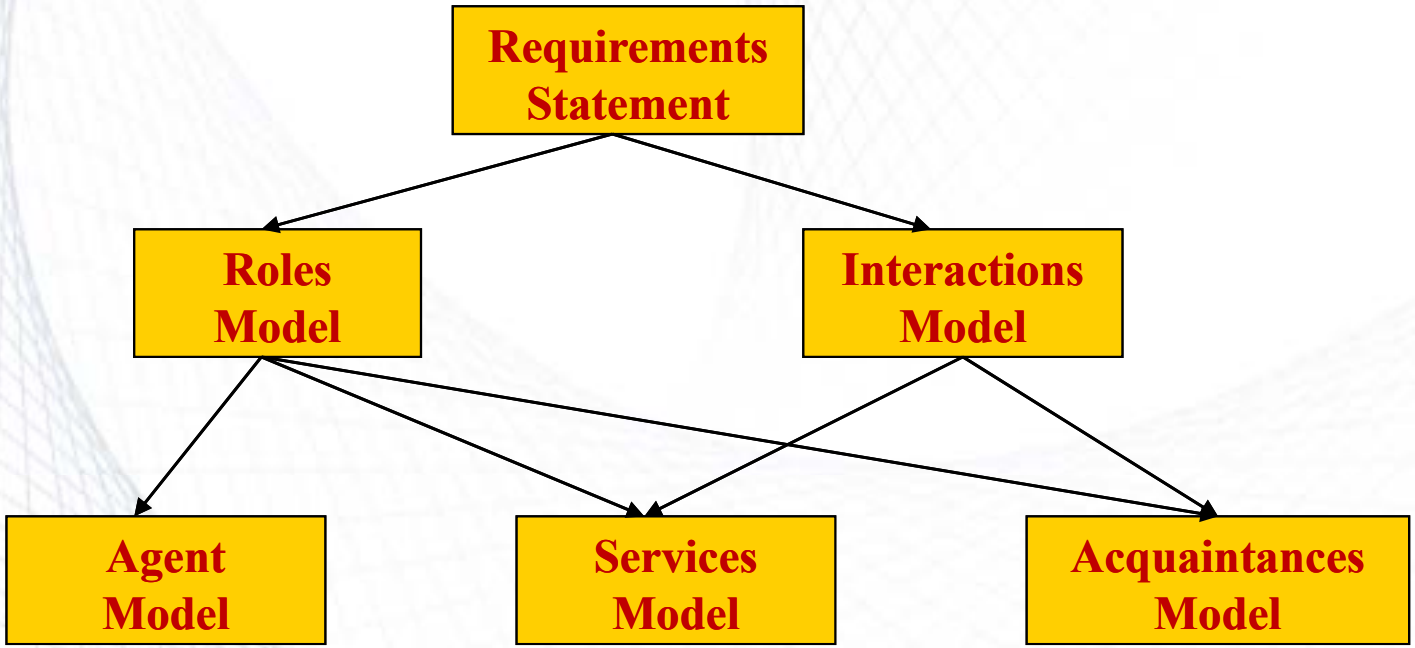


2) Design Phase

- The aim of the design phase is to transform the analysis models into a lower level of abstraction that traditional design techniques (e.g. object-oriented techniques) may be applied.
- How an agent realizes its services and how it is implemented is beyond the scope of the methodology, and will depend on the particular application domain.

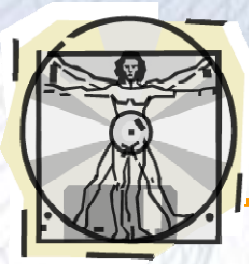


Analysis & Design Models



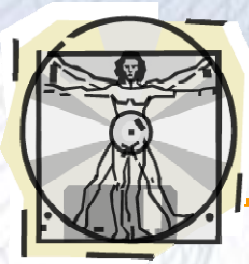
Analysis

Design →detailed analysis



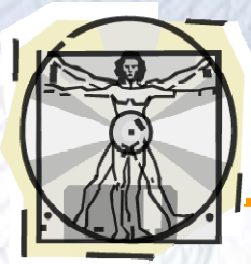
Design Models

- The design process involves generating three models.
- ***Agent model:*** identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types.
- ***Service (function or operation) model:*** identifies the main services that will be associated with each agent type.
- ***Acquaintance (collaboration) model:*** documents the acquaintances for each agent type.

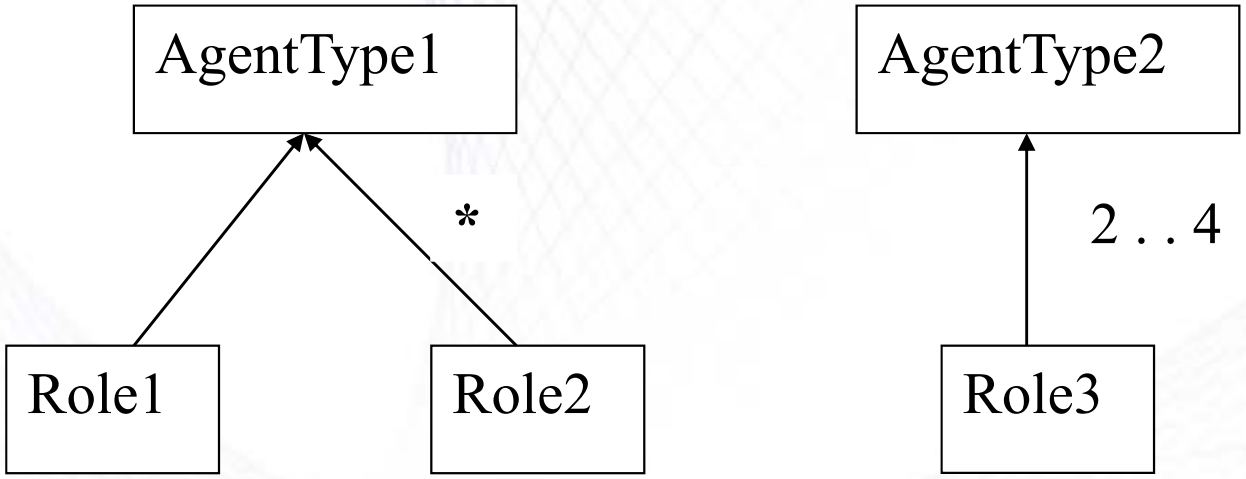


Design: Agent Model

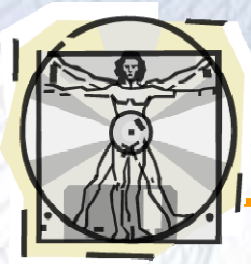
- The purpose of the ***agent model*** is to document the various agent types that will be used in the system, and the agent instances that will realise these agent types at run-time.
- An ***agent type*** is composed of a set of ***agent roles***.
- There may in fact be a one-to-one correspondence between roles and agent types. However, this need not be the case. A designer can choose to package a number of closely related roles in the same agent type for the purposes of convenience.
- The agent model is defined using a simple tree, in which root nodes correspond to agent types and leaf nodes correspond to roles, (as defined in the roles model).



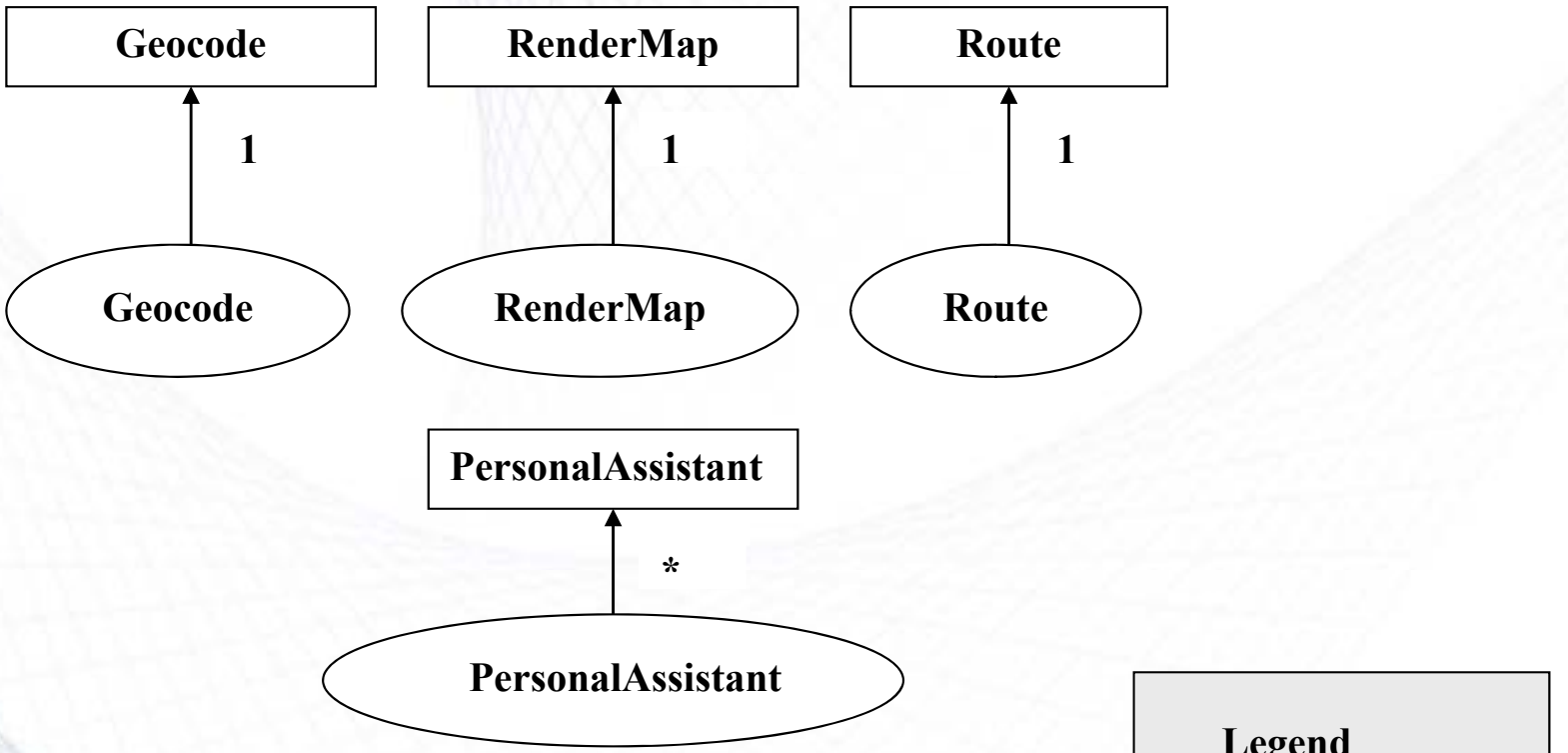
Agent Model

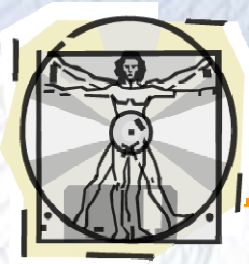


Qualifier	Meaning
n	there will be exactly n instances
m . . n	there will be between m and n instances
*	there will be 0 or more instances
+	there will be 1 or more instances



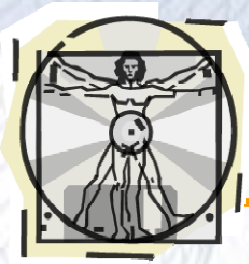
Example: Agent Model





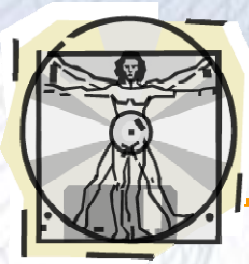
Design: Service Model /1

- The aim of the ***services model*** is to identify the services associated with each agent role, and to specify the main properties of these services.
- Service means a function or an operation of the agent. A service is a single, coherent block of activity that an agent will engage in.
- In OO terms, a service would correspond to an operation (a method); however, the services may not be available for other agents in the same way that an object's methods are available for another object to invoke.



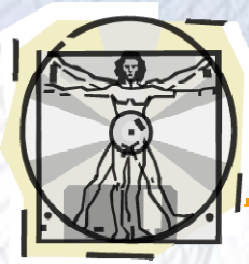
Design: Service Model /2

- For each service that may be performed by an agent, it is necessary to document its properties. Specifically, we must identify the *inputs*, *outputs*, *pre-conditions*, and *post-conditions* of each service.
- ***Inputs*** and ***outputs*** to services will be derived from the protocols model.
- ***Pre-conditions*** and ***post-conditions*** represent constraints on services. These are derived from the safety properties of a role. By definition, each role will be associated with at least one service.
- The services that an agent will perform are derived from the list of protocols and responsibilities associated with a role, and in particular, from the liveness definition of a role.



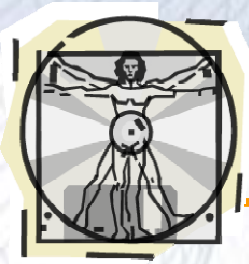
Example: Service Model

Service	*Inputs*	*Outputs*	*Pre-conditions*	*Post-conditions*
`Geocode`	Address	Latitude and Longitude of the Address	A personalized assistant agent is created and associated with the user	User enters an address
`RenderMap`	List of geocodes	URL to the map	A personalized assistant agent is created and associated with the user. The system does the proximity calculation	
`Route`	Breakdown and service provider Location	Driving direction	A personalized assistant agent is created and associated with the user	User selects the service provider

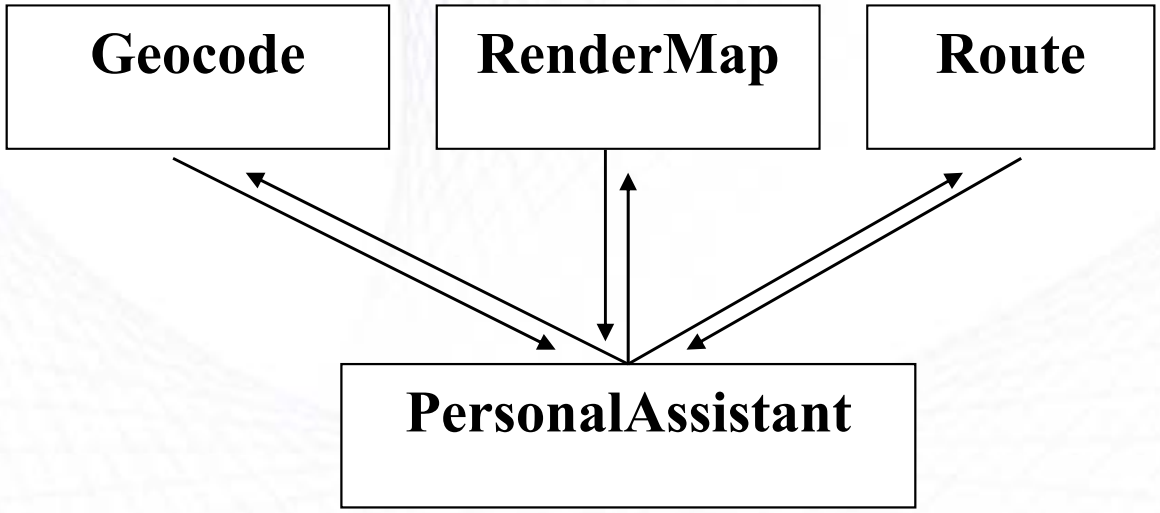


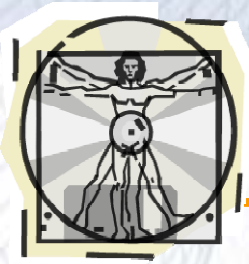
Design: Acquaintance Model

- ***Acquaintance models*** define the communication links that exist between agent types. They do not define what messages are sent or when messages are sent. They indicate that communication pathways exist.
- The purpose of acquaintance model is to identify any potential communication bottlenecks, which may cause problems at run-time. ← helpful when testing the system
- An agent acquaintance model is a graph, with nodes in the graph corresponding to agent types and arcs in the graph corresponding to communication pathways.
- Agent acquaintance models are directed graphs: **A** → **B** indicates that **A** will send messages to **B**, but not necessarily that **B** will send messages to **A**.



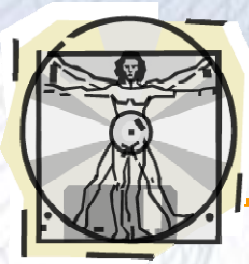
Example: Acquaintance Model





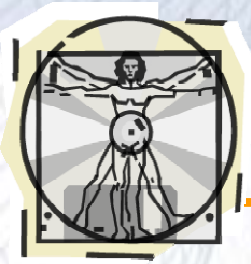
Design Process

1. Create an agent model:
   - Aggregate roles into agent types, and refine to form an agent type hierarchy;
   - Document the instances of each agent type using instance annotations.
2. Develop a services model, by examining protocols and safety and liveness properties of roles.
3. Develop an acquaintance model from the interaction model and agent model.



Extensions (Future Works)

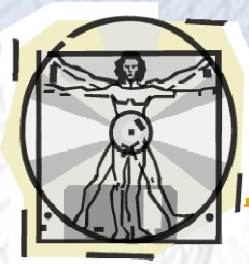
- Dealing with truly open systems, in which agents may not share common goals (← liveness property). This class of systems represents the most important application area for multi-agent systems, and it is therefore essential that the methodology be able to deal with it. ← case of competition
- Notion of an organizational structure: at the moment, such structures are only implicitly defined within the methodology via the role, interaction and acquaintance models. However, direct, explicit representations of such structures will be of value. Representing such structures may be the only way of adequately capturing and understanding the organization's communication and control structures.



Critics

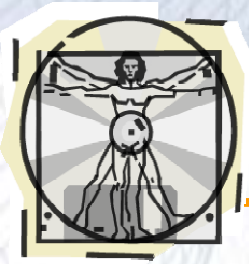
Evaluate the methodology along with the followings:

- Roles vs. tasks — ***yes***
- Emergent behaviour — ***n/a***
- Knowledge cycle — ***no***
- Interactions vs. intra-actions — ***yes***
- Symbol level communication — ***no (may be)***
- Knowledge completeness — ***no***
- Indeterminism: Decision making — ***no***



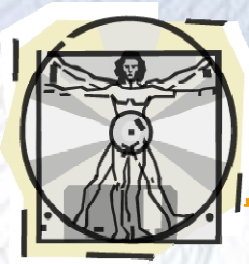
Evaluation /1

- **Concepts:** abstract and concrete concepts are clearly defined. (Good)
- **Modeling constructs:** uses a clear and consistent notation with a clear and defined syntax and semantics. None of the abstracts and concrete concepts or the models are in conflict and they can be interpreted unambiguously. (Good)
- **Scalability:** Roles are not hierarchical. The role model provides strictly one level of abstraction for the developers to conceptualize the system. The methodology does not facilitate iterative refinement of the system through different levels of abstraction. As a result, GAIA does not scale up well to handle complex systems. (Poor)



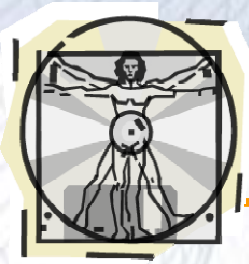
Evaluation /2

- GAIA does not have the ability to track dependencies between different models and the underling code used to implement them.
- GAIA does not directly deal with particular modeling techniques or implementation details.
- GAIA models do not specify what tasks are done in parallel, so it has a weak support for concurrency.
- GAIA offers no mechanisms to model the dynamic reasoning, extension and modification of the social aspects at runtime.
- GAIA assumes complete specification of requirements and does not address the requirement gathering phase.



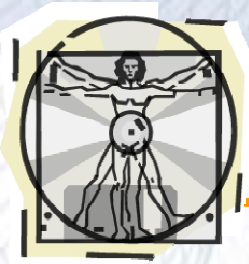
Evaluation /3

- **Limited process life-cycle coverage:** GAIA is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly, but ignoring implementation, testing/debugging, deployment and maintenance details.
- **Limited process management:** GAIA does not address any management and/or cost estimation aspects involved in software development.



Evaluation /4

- **Formal semantics:** GAIA does not have a formal semantics. A successful methodology is one that is not only of pragmatic value, but one that also has a well-defined, unambiguous formal semantics. While the typical developer need never even be aware of the existence of such a semantics, it is nevertheless essential to have a precise understanding of what the concepts and terms in a methodology mean.



Conclusions

- GAIA is a "popular" methodology for analysis and design of agent-based systems.
- The key concepts in this methodology are ***roles***, which are associated with ***responsibilities***, ***permissions***, and ***protocols***. Roles can interact with one another in certain institutionalised ways, which are defined in the protocols of the respective roles.